DEVOPS E INTEGRAÇÃO CONTÍNUA

AULA 4

Prof. Mauricio Antonio Ferste

CONVERSA INICIAL

Bem-vindo à sua jornada empolgante pelo universo dos *containers*, nesta etapa dedicada ao *Containers: Introdução ao Docker*. Ao explorar este conteúdo, você será guiado por conceitos fundamentais, com destaque para a ferramenta Docker e suas aplicações inovadoras. Vamos aprofundar a compreensão das diferenças entre *containers* e máquinas virtuais, destacando como os *containers* oferecem uma abordagem mais leve e eficiente para o desenvolvimento e a implementação de aplicações. Dedicaremos também uma seção à orquestração de *containers* com o Kubernetes, uma plataforma robusta e amplamente utilizada para gerenciar ambientes complexos de containers. Em seguida, exploraremos como os *containers* possibilitam escalabilidade e distribuição, transformando a maneira como concebemos, desenvolvemos e operamos *software* em ambientes modernos. Prepare-se para desbravar as potencialidades dessa tecnologia que está moldando o futuro da computação em nuvem e revolucionando a forma como entregamos soluções inovadoras. Este capítulo será seu guia confiável nesse fascinante mundo dos containers.

Figura 1 – Tela de Abertura do Docker

Fonte: Docker, 2023.

TEMA 1 – MÁQUINAS VIRTUAIS E SUA UTILIDADE EM DEVOPS

Este é um ponto crítico na evolução de DevOps, pois quando Patrick Debois trouxe o uso de máquinas virtuais para o contexto de desenvolvimento, uniu dois mundos diferentes, o de desenvolvimento e o de infraestrutura, pois flexibilizou a evolução de ambientes para testes e implantação. Este é certamente um momento fundamental e de virada para a instituição do DevOps como conceito.

1.1 Máquinas virtuais

Máquinas virtuais (VMs) são ambientes virtuais completamente isolados e independentes, criados para emular a funcionalidade de uma máquina física. Elas são uma forma de virtualização, que permite a execução de vários sistemas operacionais e aplicativos em um único *hardware* físico. O conceito de máquinas virtuais é fundamental para a eficiência na gestão de recursos de computação e para a criação de ambientes de teste, desenvolvimento e produção mais flexíveis.

A principal característica das máquinas virtuais é a capacidade de executar um sistema operacional completo dentro de outro, chamado de sistema hospedeiro. Essa abstração é possibilitada por um *software* conhecido como hipervisor ou monitor de máquinas virtuais. O hipervisor gerencia e aloca os recursos do sistema físico, permitindo que várias VMs coexistam independentemente (Silberschatz; Galvin, 2001).

Existem dois tipos principais de máquinas virtuais:

- **Máquinas Virtuais de Tipo 1 (*Bare Metal*)**: essas VMs são executadas diretamente sobre o *hardware*, sem a necessidade de um sistema operacional hospedeiro. O hipervisor age como um sistema operacional dedicado para gerenciar a execução das VMs. Isso resulta em melhor desempenho e eficiência, sendo comumente utilizado em ambientes de produção e servidores;
- **Máquinas Virtuais de Tipo 2 (Hospedadas)**: essas VMs são executadas sobre um sistema operacional hospedeiro convencional. Um aplicativo de virtualização é responsável por gerenciar as VMs, e o sistema operacional hospedeiro executa tarefas essenciais. Embora sejam mais fáceis de

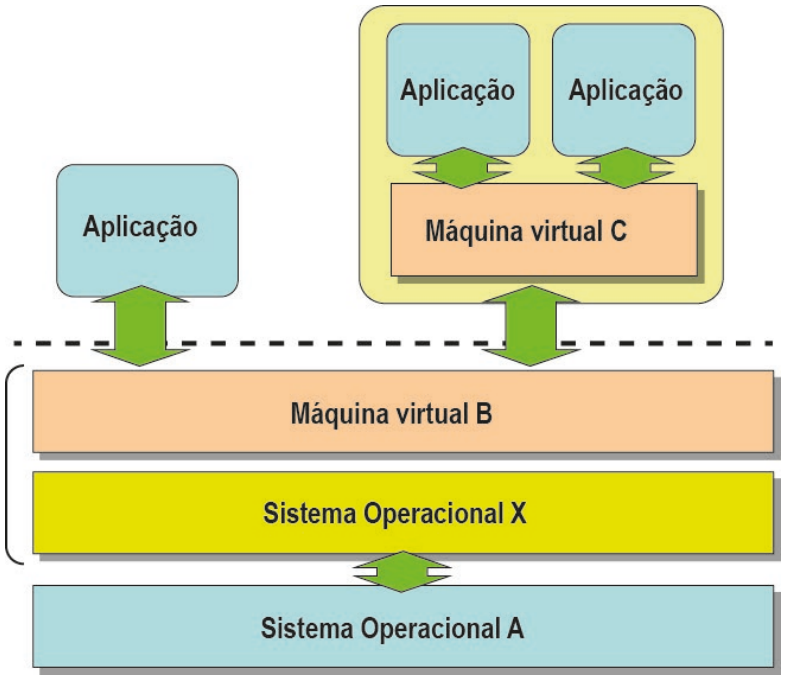
configurar e usar, as VMs de Tipo 2 geralmente possuem um desempenho inferior em comparação com as de Tipo 1 e são mais adequadas para ambientes de desenvolvimento e testes.

As máquinas virtuais oferecem diversos benefícios, tais como:

- **Isolamento**: cada VM opera independentemente das outras, garantindo isolamento completo. Isso é crucial para segurança e estabilidade;
- **Flexibilidade**: a capacidade de executar diferentes sistemas operacionais em um único hardware oferece flexibilidade significativa para testes, desenvolvimento e implantação de aplicativos;
- **Consolidação de Servidores**: várias VMs podem ser executadas em um único servidor físico, otimizando a utilização dos recursos e reduzindo a necessidade de *hardware* físico;
- **Recuperação de Desastres**: VMs podem ser facilmente copiadas, movidas ou restauradas, tornando a recuperação de desastres mais rápida e eficiente.

As máquinas virtuais desempenham um papel fundamental na virtualização, que é uma tecnologia amplamente adotada em ambientes empresariais para melhorar a eficiência operacional e a flexibilidade do gerenciamento de recursos de TI.

Figura 2 – Visão geral das diferentes camadas de um sistema computacional



Crédito: Arte/UT.

Considere a possibilidade de provisionar uma máquina virtual em questão de segundos, sem nenhum custo associado, ao contrário do que aconteceria ao aguardar dias ou até mesmo semanas para configurar um novo ambiente físico. É claro que não é viável executar um número infinito de VMs em um único hospedeiro, mas a utilização de virtualização pode, em determinadas circunstâncias, desvincular a necessidade de adquirir *hardware* do ciclo de vida dos ambientes em que essas máquinas virtuais operam.

Saiba mais

Acesse o *link* a seguir e tenha a uma visão sobre a evolução do uso de ambientes virtuais em DevOps.

ZAMORANO, H. M. Virtualização de ambientes em DevOps. **HNZ**, 11 maio 2021. Disponível em: <https://hnz.com.br/virtualizacao-de-ambientes-em-devops/>. Acesso em: 16 abr. 2024.

Então, retomando, as máquinas virtuais (VMs) são uma tecnologia que permite executar vários sistemas operacionais independentes em um único *host* físico. Isso é possível graças à virtualização, que é um processo que abstrai o *hardware* físico e o apresenta aos sistemas operacionais como uma máquina virtual. Vamos ver agora algumas características.

- **Processos e chaveamento de contexto:** as VMs são uma extensão do conceito de processo. Um processo é uma abstração que representa um programa em execução. Ele tem seu próprio espaço de endereçamento lógico, seus próprios registradores lógicos e seu próprio estado de execução. O chaveamento de contexto é um mecanismo que permite ao sistema operacional alternar entre processos. Ele é essencial para simular a execução simultânea de vários processos;
- **Estrutura hierárquica dos sistemas operacionais:** os sistemas operacionais são sistemas complexos que são compostos de vários componentes. Esses componentes são organizados em uma estrutura hierárquica, com diferentes níveis de abstração e interfaces bem definidas. As interfaces são essenciais para a independência entre componentes de *hardware* e *software*. Elas permitem que cada componente seja visto como um subsistema independente.

Implementação de máquinas virtuais:

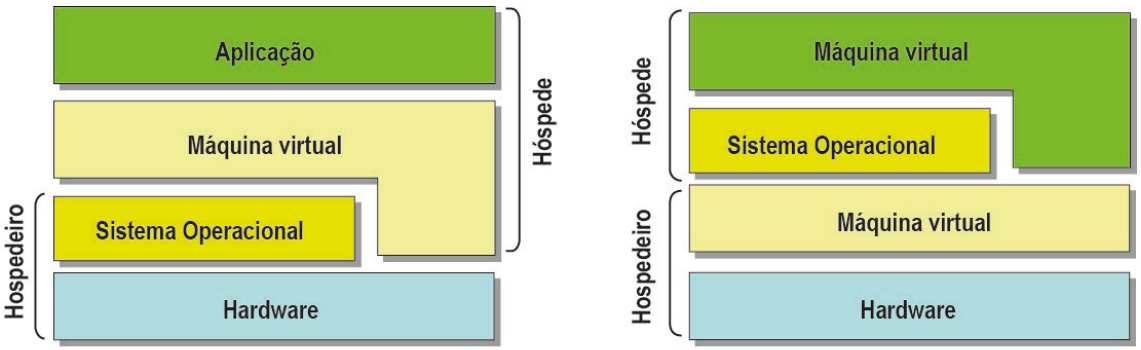
Existem duas abordagens principais para a implementação de máquinas virtuais:

- **Máquinas virtuais de processo**: um programa de aplicação fornece um ambiente de execução para outras aplicações. As máquinas virtuais de processo são criadas por um programa de aplicação. Elas são limitadas ao espaço de endereçamento do programa de aplicação que as criou;
- **Máquinas virtuais de sistema**: uma camada de *software* (hipervisor) atua como intermediário entre o *hardware* e o sistema operacional. As máquinas virtuais de sistema são criadas por um hipervisor. Elas têm acesso total ao *hardware* físico.

Diferenças entre máquinas virtuais de processo e máquinas virtuais de sistema

A principal diferença entre máquinas virtuais de processo e máquinas virtuais de sistema é que as máquinas virtuais de sistema têm acesso total ao *hardware* físico, enquanto as máquinas virtuais de processo são limitadas ao espaço de endereçamento do programa de aplicação que as criou.

Figura 3 – Arquitetura de máquinas virtuais



Crédito: Arte/UT.

A Figura 3 esquematiza as diferenças entre máquinas virtuais de processo e máquinas virtuais de sistema.

As máquinas virtuais são uma tecnologia poderosa que pode ser usada para uma variedade de propósitos. Elas permitem executar vários sistemas operacionais independentes em um único *host* físico, o que pode simplificar a administração de sistemas e aumentar a flexibilidade.

TEMA 2 – CONTAINERS: INTRODUÇÃO AO DOCKER

Os contêineres têm revolucionado a maneira como desenvolvemos, distribuímos e executamos aplicativos. Nesse contexto, o Docker se destaca como uma das principais ferramentas para facilitar a criação e gerenciamento desses ambientes isolados. Vamos explorar uma introdução ao Docker para compreender melhor como essa tecnologia está transformando a forma como lidamos com aplicações.

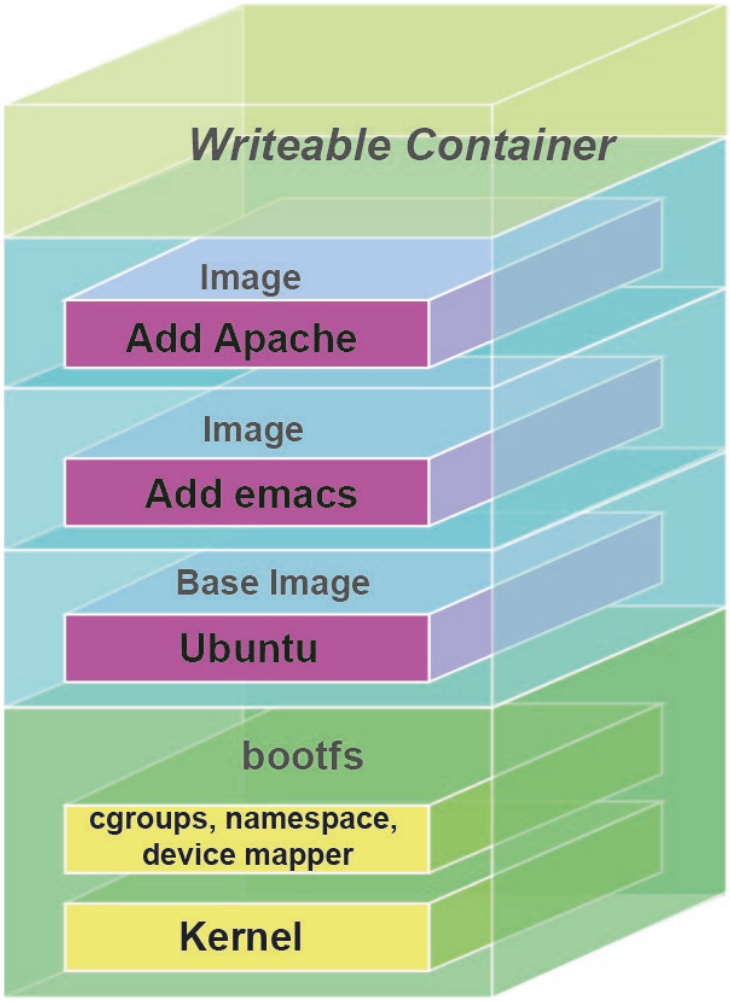
2.1 *Container*

Contêineres são ambientes leves e isolados que encapsulam uma aplicação e suas dependências. Eles proporcionam consistência entre diferentes ambientes de desenvolvimento, teste e produção, eliminando as divergências que podem surgir devido a diferenças nos sistemas operacionais ou configurações.

Um contêiner representa um ambiente isolado para o seu código, caracterizado pela ausência de conhecimento sobre o sistema operacional ou os arquivos do *host*. Sua execução ocorre no ambiente fornecido pelo Docker Desktop. Os contêineres são autossuficientes, contendo tudo o que é necessário para a execução do seu código, inclusive um sistema operacional básico. O Docker Desktop oferece a capacidade de gerenciar e explorar seus contêineres de forma conveniente, para entender isso temos de entender um sistema.

Vamos entender, como na Figura 4 como o sistema operacional funciona como um intermediário entre o *hardware* e os aplicativos. O *kernel* gerencia os recursos do *hardware* e fornece serviços básicos para os aplicativos. Os aplicativos interagem com o *kernel* através de APIs, o que permite que eles sejam executados de forma independente do *hardware* específico.

Figura 4 – Arquitetura de um sistema



Fonte: Terkaly, 2016.

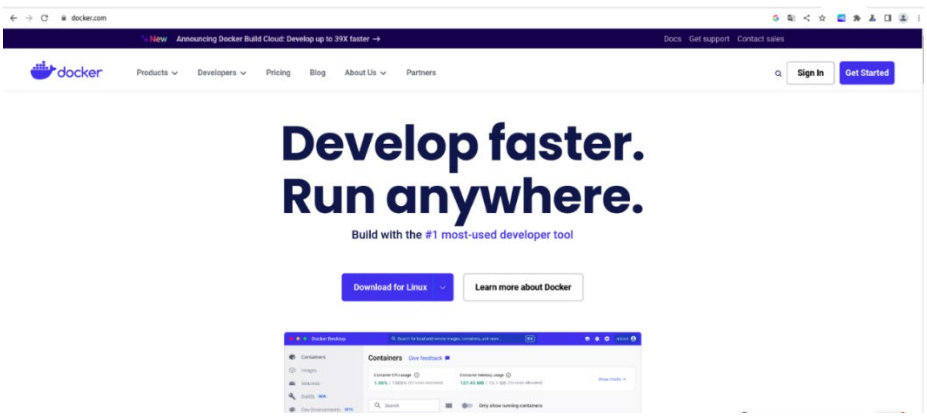
Ou seja, como visto na Figura 4, dentro de um contêiner Docker, a estrutura do sistema operacional é organizada em três camadas principais: *hardware*, *kernel* e aplicativos. A camada de *hardware* representa os recursos físicos disponibilizados pelo *host*, como CPU, memória e dispositivos de E/S, que são compartilhados com o contêiner. O *kernel*, que atua como o núcleo do sistema operacional, é encapsulado dentro do contêiner, gerenciando os recursos fornecidos pelo *host* e oferecendo serviços essenciais para os aplicativos contidos no contêiner. Os Aplicativos são os programas em execução dentro do contêiner, interagindo com o *kernel* por meio de APIs específicas, o que permite sua portabilidade e execução independente do ambiente de *hardware* subjacente. Esta estrutura simplificada demonstra o papel intermediário do sistema operacional em um ambiente de conteinerização, onde as camadas são isoladas e os recursos são compartilhados de forma controlada, facilitando a implantação e distribuição de aplicativos.

2.2 Docker

Uma imagem do Docker consiste em sistemas de arquivos sobrepostos, sendo a base um sistema de arquivos de inicialização, chamado *bootfs*, semelhante ao típico sistema de arquivos de inicialização do Linux/Unix. Normalmente, os usuários do Docker não interagem diretamente com o sistema de arquivos de inicialização. Após a inicialização de um contêiner, ele é movido para a memória e o sistema de arquivos de inicialização é desmontado para liberar a RAM usada pela imagem do disco (Docker, 2023).

Neste ponto, a configuração se assemelha a uma pilha de virtualização Linux convencional. No entanto, o Docker vai além ao sobrepor um sistema de arquivos raiz, conhecido como *rootfs*, sobre o sistema de arquivos de inicialização. Esse *rootfs* pode representar um ou mais sistemas operacionais, como um sistema de arquivos Debian ou Ubuntu. Nos cenários tradicionais de inicialização do Linux, o sistema de arquivos raiz é montado como somente leitura e depois alterado para leitura-escrita após a inicialização, seguindo uma verificação de integridade. No entanto, no mundo do Docker, o sistema de arquivos raiz permanece em modo somente leitura. O Docker utiliza um *union mount* para adicionar sistemas de arquivos adicionais somente leitura ao sistema de arquivos raiz. Um *union mount* permite que vários sistemas de arquivos sejam montados ao mesmo tempo, mas apareçam como um único sistema de arquivos. Ele sobreposiciona os sistemas de arquivos uns sobre os outros, criando um sistema de arquivos resultante que pode conter arquivos e subdiretórios de qualquer um dos sistemas de arquivos subjacentes (Izidório, 2019).

Figura 5 – Tela inicial do sítio do Docker, sempre procurar entender a arquitetura no sítio formal da aplicação.



Fonte: Docker, 2023.

Cada um desses sistemas de arquivos no Docker é chamado de *imagem*, e essas imagens podem ser sobrepostas umas sobre as outras. A imagem mais abaixo é chamada de *imagem pai*, e você pode percorrer cada camada até chegar à imagem base na parte inferior da pilha. Ao lançar um contêiner a partir de uma imagem, o Docker monta um sistema de arquivos de leitura-escrita sobre quaisquer camadas subjacentes. Este sistema de arquivos de leitura-escrita é onde os processos desejados para o contêiner Docker serão executados.

2.2.1 Comandos de Contêiner e Docker

Você usa comandos com parâmetros para manipular contêineres do Docker. O formato de comando padrão é

```
docker [opções] [comando] [argumentos].
```

A seguir contém comandos de contêiner usados com frequência. Há vários outros listados na documentação do Docker.

```
docker ps -a
```

Lista todos os contêineres. O sinalizador -a mostra contêineres em execução e não em execução. Para exibir somente contêineres em execução, esse sinalizador pode ser omitido.

```
docker rename [contêiner] [novo_nome]
```

Renomeia o contêiner fornecido como novo_nome.

```
docker start [contêiner]
```

Executa o contêiner fornecido.

```
docker stop [contêiner]
```

Interrompe o contêiner fornecido.

```
docker wait [contêiner]
```

Faz com que o contêiner especificado espere até que outros contêineres em execução sejam interrompidos.

Comandos de imagem

Há menos comandos de imagem em comparação aos comandos de contêiner.

```
Docker build -t image_name .
```

Cria uma imagem do Docker com a tag image_name a partir dos arquivos no diretório atual.

```
docker create [imagem]
```

Cria um contêiner não executado a partir da imagem fornecida.

```
docker run [imagem]
```

Cria e executa um contêiner com base na imagem fornecida.

TEMA 3 – CONTAINERS *VS* MÁQUINAS VIRTUAIS

A questão de se containers são máquinas virtuais é frequentemente levantada quando se discute containers. Neste artigo, exploraremos conceitos fundamentais que diferenciam containers de máquinas virtuais (VMs), focando especificamente em containers utilizando o Docker. O Docker, desenvolvido pela empresa de *software* de mesmo nome, é um sistema de gerenciamento de containers de código aberto, disponível nas versões Community e Enterprise. Fundada em 2013, a empresa inicialmente era chamada de DotCloud e liberou os códigos fonte, escritos em Golang (Go), uma linguagem de programação, que foi desenvolvido dentro da Google. A tecnologia Docker é amplamente adotada por diversas empresas. Com uma comunidade ativa, comprometida com o desenvolvimento contínuo, ajustes, testes e correção de falhas, o Docker é conhecido por sua leveza, portabilidade e natureza híbrida. Estas características serão exploradas ao longo do artigo, proporcionando uma visão abrangente sobre containers e suas vantagens em comparação com a infraestrutura de virtualização convencionalmente utilizada para fornecer serviços.

3.1 Contextualização

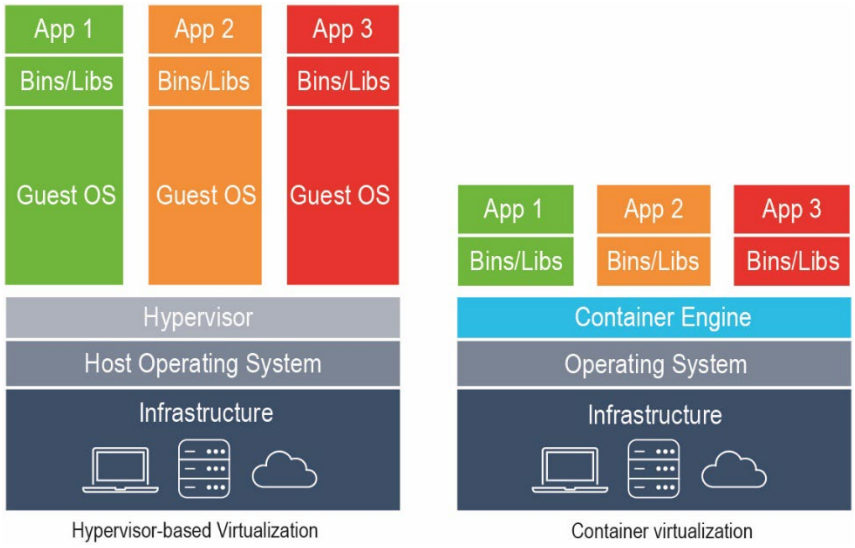
A função primordial de um *container* é oferecer micro serviços, eliminando a necessidade de instalar um sistema operacional completo para o funcionamento adequado de um serviço específico. Isso resulta em maior agilidade ao implementar novos serviços, como um servidor *web* utilizando Apache ou Tomcat, por exemplo. Containers constituem ambientes isolados dentro de um sistema operacional, sendo este último obrigatoriamente 64 bits e podendo ser Windows, Linux ou Mac OS. A técnica de isolamento de ambientes, intrínseca aos containers, tem origens que remontam a 1979 com a utilização do isolamento de processos no sistema operacional UNIX por meio do *chroot*,

conforme representado na linha do tempo da evolução dos *containers* na imagem abaixo.

Ao contrário de uma máquina virtual, que possui seu próprio *kernel*, binários e bibliotecas, o Docker compartilha o *kernel*, binários e bibliotecas do sistema operacional no qual está sendo executado. Essas diferenças são claramente delineadas na imagem subsequente, evidenciando as distinções entre máquinas virtuais e *containers* (Terkaly, 2016).

Já na Figura 6, temos as diferentes camadas de abstração em uma máquina virtual, desde o *hardware* até o *software* aplicativo. Está dividido em cinco camadas principais. A primeira camada, *hardware*, representa os componentes físicos da máquina virtual, como CPU, memória e dispositivos de entrada/saída, essenciais para o funcionamento do sistema operacional. A segunda camada, *firmware*, refere-se ao *software* presente na memória ROM, responsável pela inicialização da máquina virtual e pelo carregamento do sistema operacional. Na terceira camada, Sistema Operacional, o *kernel* gerencia os recursos de hardware e fornece serviços básicos aos aplicativos. O *middleware*, na quarta camada, age como intermediário entre o sistema operacional e os aplicativos, oferecendo serviços como gerenciamento de transações e segurança. Por fim, na quinta camada, estão os Aplicativos, como editores de texto e navegadores *web*, que interagem com o *kernel* por meio de APIs. Este diagrama simplificado ilustra a estrutura de uma máquina virtual, destacando as camadas de abstração e suas respectivas funções no contexto virtualizado.

Figura 6 – Notícias



Fonte: Terkaly, 2016.

Co*n*tainers Docker são ambientes de execução isolados que compartilham o *kernel* do sistema operacional do *host*, tornando-os mais leves e portáteis do que máquinas virtuais, que necessitam de seu próprio sistema operacional completo. A portabilidade dos containers Docker é viabilizada pelo Docker Engine, uma camada de abstração que gerencia esses ambientes. O Docker Engine é executado no *host*, proporcionando uma interface de linha de comando (CLI) para a criação, inicialização e administração dos containers. Outro destaque dos containers Docker é a capacidade de criar imagens customizadas, que são arquivos contendo tudo necessário para executar um container, incluindo código, bibliotecas e configurações. Essa flexibilidade permite que os desenvolvedores criem *containers* com configurações precisas.

As imagens Docker podem ser compartilhadas no Docker Hub, um repositório público, facilitando a reutilização e a criação de novos containers a partir de existentes. Containers Docker podem ser orquestrados por *softwares* especializados, permitindo que administradores gerenciem um grande número de *containers*, distribuam a carga de trabalho, garantam alta disponibilidade e escalem conforme necessário (Terkaly, 2016).

A capacidade de orquestração é uma das principais vantagens dos containers Docker em relação às máquinas virtuais, proporcionando gerenciamento centralizado e escalabilidade eficiente, o que pode reduzir custos e aumentar a eficiência. Resumindo, os containers Docker representam uma tecnologia de virtualização com diversas vantagens sobre as máquinas virtuais, sendo leves, portáteis, fáceis de gerenciar e escaláveis. Erros comuns sobre containers Docker incluem confundir *containers* com máquinas virtuais, subestimar as vantagens e desvantagens dos containers em relação às máquinas virtuais, e pensar que *containers* só podem ser executados em máquinas virtuais, quando, na verdade, podem ser executados diretamente em *hardware* físico.

TEMA 4 – ORQUESTRAÇÃO COM KUBERNETES

A orquestração com Kubernetes envolve a administração eficiente de containers em ambientes distribuídos. Utilizando uma abordagem baseada em YAML (que veremos adiante) para definir configurações, os usuários podem criar e gerenciar *pods*, que são conjuntos de containers, e serviços, que oferecem funcionalidades de rede. O Kubernetes facilita a escalabilidade automática, a

gestão de estado e a distribuição de carga, permitindo aos administradores lidarem com aplicativos complexos de forma centralizada. Com recursos como o Kubernetes Deployment, os usuários podem atualizar aplicações sem tempo de inatividade, enquanto o sistema automatiza a recuperação de falhas. Essa plataforma robusta simplifica a orquestração de containers, proporcionando flexibilidade e escalabilidade em ambientes de produção.

4.1 YAML

A YAML (YAML Ain't Markup Language) é um formato de serialização de dados legível por humanos e de fácil escrita. Comumente utilizado para configurações e descrições de dados hierárquicos, o YAML utiliza espaçamento para definir a estrutura do documento, dispensando o uso de caracteres especiais, como colchetes e parênteses. Sua sintaxe simples e intuitiva o torna amplamente adotado em configurações de software, incluindo a orquestração de containers com ferramentas como Kubernetes. No contexto do Kubernetes, os arquivos YAML são frequentemente empregados para definir configurações de pods, serviços, volumes e outros recursos, proporcionando uma maneira concisa e legível de descrever o estado desejado do sistema.

Saiba mais

Se desejar saber mais acesse o site formal de YAML em:

YAML. Disponível em: <https://yaml.org/>. Acesso em: 16 abr. 2024.

De certa forma a linguagem pode ser substituída por JSON, embora seja muito mais legível para o humano, veja abaixo uma comparação:

```
# YAML
AWSTemplateFormatVersion: '2023-09-09'
Resources:
  myStack:
    Type: AWS::CloudFormation::Stack
    Properties:
      TemplateURL: https://s3.amazonaws.com/cloudformation-
templates-us-east-1/S3_Bucket.template
      TimeoutInMinutes: '60'
Outputs:
  StackRef:
    Value: !Ref myStack
  OutputFromNestedStack:
    Value: !GetAtt myStack.Outputs.BucketName
```

Observe que a linguagem YAML é mais legível que o JSON abaixo:

```
# JSON
{
    "AWSTemplateFormatVersion" : "2023-09-09",
    "Resources" : {
        "myStack" : {
            "Type" : "AWS::CloudFormation::Stack",
            "Properties" : {
               "TemplateURL" :
"https://s3.amazonaws.com/cloudformation-templates-us-east-
1/S3_Bucket.template",
              "TimeoutInMinutes" : "60"
            }
        }
    },
    "Outputs": {
       "StackRef": {"Value": { "Ref" : "myStack"}},
       "OutputFromNestedStack" : {
             "Value" : { "Fn::GetAtt" : [ "myStack",
"Outputs.BucketName" ] }
        }
    }
}
```

YAML é uma linguagem de marcação humanamente legível que é frequentemente usada para descrever configurações e dados. É uma linguagem de sintaxe simples e fácil de aprender, mas existem algumas regras básicas que você deve seguir ao criar arquivos YAML.

Regras para criar um arquivo YAML

Ao criar um arquivo YAML, observe as seguintes regras fundamentais:

YAML faz distinção entre maiúsculas e minúsculas. Isso significa que "chave" é diferente de "Chave".

Os arquivos devem ter a extensão. yaml. Isso ajuda a distinguir arquivos YAML de outros tipos de arquivos.

Use apenas espaços em arquivos YAML.

Os componentes básicos do YAML incluem:

- Mapas: Mapas são usados para armazenar dados associativos, como chaves e valores;
- Listas: listas são usadas para armazenar uma sequência de dados;
- Strings: *strings* são usadas para armazenar dados de texto;
- Números: números são usados para armazenar dados numéricos;

- Booleanos: booleanos são usados para armazenar dados lógicos, como verdadeiro ou falso.

Mapas: são usados para armazenar dados associativos, como chaves e valores. Chaves são usadas para identificar valores, e valores podem ser de qualquer tipo de dados. Para criar um mapa, use a seguinte sintaxe:

```
nome: João
idade: 30
```

Listas: armazenam uma sequência de dados. Os dados podem ser de qualquer tipo, incluindo mapas, listas, *strings*, números e booleanos. Segue:

```
produtos:
- nome: Computador
  preço: 2.000
 - nome: Celular
  preço: 1.000
```

Neste exemplo, "produtos" é uma lista que contém dois itens. Cada item é um mapa que contém um nome e um preço.

Strings: *strings* são usadas para armazenar dados de texto. *Strings* podem ser delimitadas por aspas simples (') ou aspas duplas ("). Veja abaixo:

```
nome: 'João da Silva'
```

Números: Números são usados para armazenar dados numéricos. Números podem ser inteiros, decimais ou de ponto flutuante. Veja a seguir:

```
idade: 30
```

Neste exemplo, "30" é um número inteiro.

Booleanos: Booleanos são usados para armazenar dados lógicos, como verdadeiro ou falso.

Para criar um booleano, use a seguinte sintaxe:

```
ativo: true
```

Por fim um exemplo de arquivo YAML que usa os detalhes acima:

```
nome: João da Silva
idade: 30
produtos:
- nome: Computador
  preço: 2.000
```

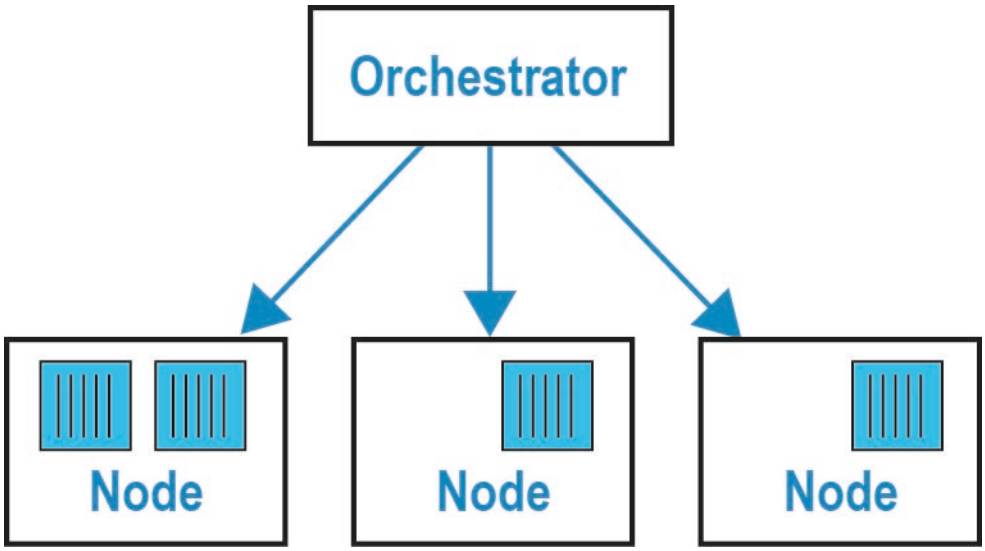
```
- nome: Celular
  preço: 1.000
ativo: true
```

Evidente que, por se tratar de um estudo mais amplo estamos tratando de explanar os detalhes principais, mas um curso aprofundado para quem deseja tratar especificamente pode ser necessário.

4.2 Gerenciamento de configuração na teoria

Em uma escala mais ampla, as equipes precisam adquirir habilidades para gerenciar e padronizar as configurações do sistema. Isso envolve definir a infraestrutura como código e praticar a implementação de alterações de maneira controlada e sistemática. Ao longo do tempo, as equipes utilizam ferramentas de configuração para acompanhar o estado desejado do sistema, evitando desvios na configuração do recurso. O gerenciamento de configuração refere-se à abordagem sistemática e controlada de implementar alterações, reduzindo os riscos associados à modificação da configuração do sistema.

Figura 7 – Visão básica de um orquestrador para gerência de vários hosts Docker ou nodes



Fonte: Bashir, 2018; Crédito: Arte/UT.

4.3 Gerenciamento de configuração na prática – Kubernetes

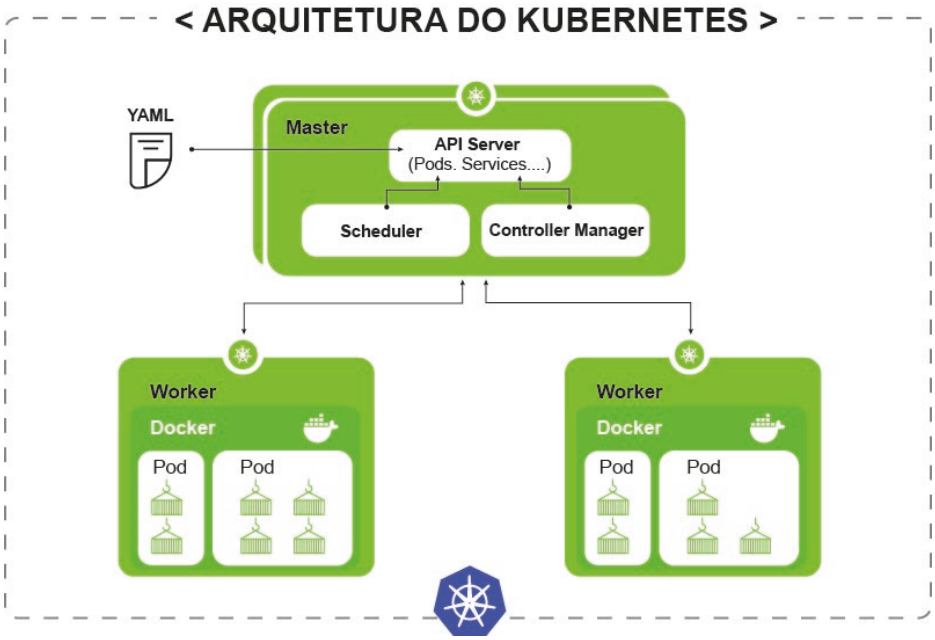
Nome em grego significa *timoneiro* ou *piloto de navio*, *Kubernetes* é um orquestrador de *containers* que simplifica o desenvolvimento e a administração de aplicações em ambientes de *containers*. Sem um orquestrador, gerenciar um

conjunto de *containers* pode ser desafiador, exigindo a administração manual de execução, dimensionamento e escalabilidade. O Kubernetes automatiza essas tarefas, gerenciando de maneira eficiente *clusters* com várias máquinas, destacando-se por sua arquitetura robusta. A estrutura do Kubernetes compreende dois componentes principais: o Mestre, responsável por gerenciar o cluster e manter seu estado desejado, e os Nós (ou trabalhadores), responsáveis pela execução das aplicações. No Mestre, três componentes essenciais se destacam: a API, encarregada de receber e executar comandos; o Scheduler, responsável por determinar onde os Pods (representando cápsulas de containers) serão executados no cluster; e o Controller Manager, que mantém o estado do cluster conforme especificado.

A API, exposta via HTTP, permite a interação entre desenvolvedores e administradores de sistemas com o Kubernetes. O Scheduler considera fatores como disponibilidade de recursos, localização dos Pods e políticas de replicação para determinar a execução dos Pods. Já o Controller Manager monitora o cluster e toma medidas corretivas quando necessário.

Kubernetes oferece vantagens como simplicidade, automatizando tarefas complexas; eficiência, gerenciando clusters de forma otimizada; e escalabilidade para atender às demandas de qualquer aplicação. É uma ferramenta poderosa, popular em empresas de todos os portes, dada sua arquitetura robusta e benefícios significativos.

Figura 8 – Arquitetura do Kubernetes



Fonte: Kubernates, 2023. Arte/UT.

Na construção de aplicações com Kubernetes, empregamos a ferramenta *kubectl*, que possibilita a comunicação com a API do Kubernetes e a realização de requisições dentro do cluster. A API do Kubernetes é uma interface que viabiliza a interação entre desenvolvedores e administradores de sistemas com o Kubernetes, sendo acessível por HTTP para a realização de requisições utilizando diversas linguagens de programação ou ferramentas que suportam HTTP. O *kubectl*, por sua vez, é uma ferramenta de linha de comando equipada com um conjunto de comandos que facilitam a criação, gestão e monitoramento de aplicações em *containers*. Para enviar as definições desejadas à API do Kubernetes por meio do *kubectl*, utilizamos arquivos em formato JSON ou YAML. Os arquivos YAML, notadamente, são o formato mais comum e de fácil manipulação. Estruturados como dicionários ou listas, os arquivos YAML são essenciais para descrever, por exemplo, a definição de um Pod, conforme ilustrado no exemplo abaixo.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: meu-pod
spec:
  containers:
  - name: meu-container
    image: nginx
```

Este arquivo define um Pod chamado "meu-pod" que contém um container chamado "meu-container". O *container* "meu-container" está usando a imagem nginx. Para criar um Pod usando o *kubectl*, podemos usar o seguinte comando:

```
kubectl create -f meu-pod.yaml
```

Este comando criará um Pod com o nome e as configurações definidos no arquivo YAML.

Outros recursos

Além de Pods, o Kubernetes também suporta outros recursos, como serviços, *deployments*, replica *sets* e *namespaces*. Esses recursos podem ser usados para criar aplicações mais complexas. O Kubernetes é uma ferramenta poderosa que pode simplificar significativamente a criação e gestão de aplicações em *containers*.

4.4 PODS

Para terminar, vamos tratar de *pods*, os *pods* representam as unidades fundamentais de computação no Kubernetes, consistindo em conjuntos de containers que compartilham espaço de rede e armazenamento. Em situações em que um *Pod* inclui múltiplos *containers*, há a possibilidade de compartilhar recursos como armazenamento, redes e bibliotecas entre eles. Essa prática facilita a comunicação e contribui para a eficiência da aplicação. Além disso, os *Pods* desempenham um papel crucial na escalabilidade, permitindo que o Kubernetes replique *Pods* para aumentar a capacidade de uma aplicação. Essa capacidade de replicação possibilita que a aplicação atenda a um aumento na demanda sem a necessidade de reimplementação.

TEMA 5 – ESCALABILIDADE E DISTRIBUIÇÃO COM CONTAINERS

A escalabilidade e distribuição com *containers* desempenham um papel crucial no panorama moderno de desenvolvimento de *software*. Containers oferecem uma abordagem flexível e eficiente para lidar com o aumento da demanda e a complexidade das aplicações. A capacidade de escalabilidade permite que as aplicações cresçam ou encolham dinamicamente em resposta às variações de tráfego, otimizando a utilização de recursos. Com a distribuição de containers, é possível implantar aplicações em ambientes diversos, proporcionando maior flexibilidade e resiliência. Organizações podem alcançar uma escalabilidade mais eficiente em comparação com abordagens tradicionais. A natureza leve e isolada dos *containers* permite a rápida criação, replicação e distribuição de unidades de aplicação, possibilitando uma resposta ágil às mudanças nas demandas do usuário. Além disso, a distribuição estratégica de containers em vários nós ou *clusters* assegura maior disponibilidade e confiabilidade, reduzindo pontos únicos de falha.

Kubernetes permite gerenciar automaticamente o dimensionamento horizontal, distribuindo a carga de trabalho entre os containers para otimizar o desempenho. Essa capacidade é essencial para atender eficazmente a picos de tráfego, garantindo uma experiência consistente para os usuários finais. Em resumo, a escalabilidade e distribuição com containers são fundamentais para enfrentar os desafios dinâmicos do cenário atual de desenvolvimento de

software, permitindo que as aplicações se adaptem rapidamente às mudanças nas demandas e ofereçam desempenho confiável em ambientes variados.

5.1 Contextualizando IAC

Infraestrutura como código (IaC) é uma abordagem para gerenciar infraestrutura usando código. Em vez de configurar manualmente a infraestrutura, as equipes usam arquivos de código para descrever os recursos e topologias de infraestrutura necessários. Esses arquivos de código podem ser armazenados e versionados em sistemas de controle de versão, o que facilita a colaboração, a revisão e a reversão de alterações. Ela oferece uma série de benefícios para as equipes de TI, incluindo redução de erros, transparência e repetibilidade, e agilidade. A IaC ajuda a reduzir erros humanos ao automatizar o processo de implantação de infraestrutura. Fornece uma descrição clara e concisa da infraestrutura, facilitando a compreensão e manutenção. Além disso, permite que as equipes implantem infraestrutura rapidamente e de forma controlada, sendo essencial para ambientes dinâmicos.

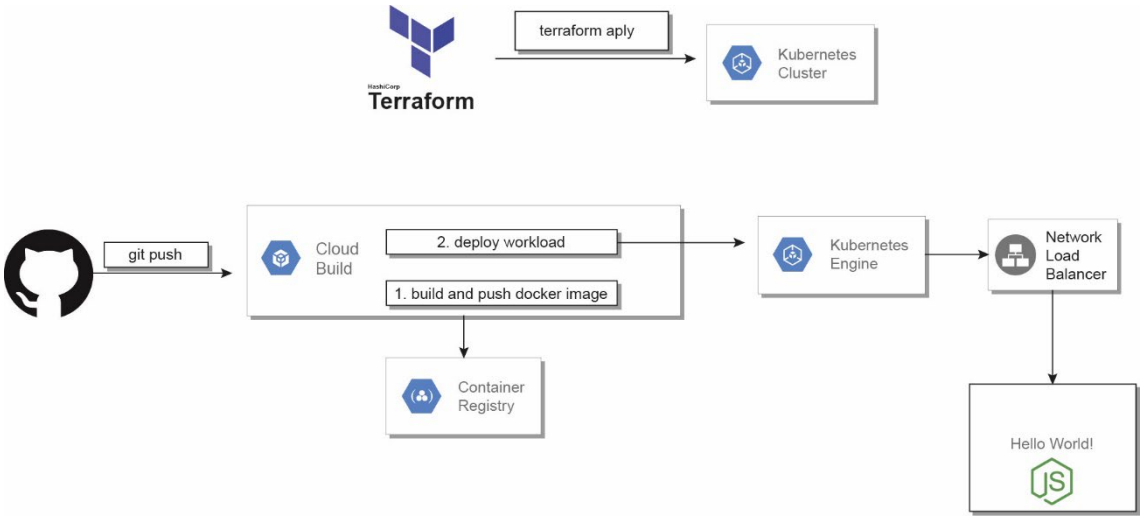
Temos um conjunto de ferramentas e técnicas para definir e implantar infraestrutura. As ferramentas de IaC mais comuns incluem YAML, que já vimos anteriormente e é um formato de arquivo de texto humanamente legível frequentemente usado para definir infraestrutura; Terraform, uma ferramenta open source para criar e gerenciar infraestrutura usando código; e AWS CloudFormation, uma ferramenta da AWS para criar e gerenciar infraestrutura na nuvem usando código (usando arquivos de configuração ou de marcação como JSON e YAML) O AWS Cloud Development Kit (AWS CDK) é uma estrutura de desenvolvimento de software que permite aos desenvolvedores definir infraestrutura como código (IaC) usando linguagens de programação familiares, como TypeScript, Python, Java e C#. Com o AWS CDK, os desenvolvedores podem criar e gerenciar recursos de infraestrutura na AWS de forma programática, usando abstrações de alto nível para definir recursos como instâncias EC2, bancos de dados RDS e buckets S3. Isso permite que as equipes apliquem práticas de desenvolvimento de *software*, como controle de versão, teste automatizado e reutilização de código, à infraestrutura em nuvem, proporcionando uma maneira mais eficiente e escalável de criar e manter recursos na AWS).

Saiba mais

AWS Cloud Development Kit. **AWS**, S.d. Disponível em: <https://aws.amazon.com/pt/cdk/>. Acesso em; 16 abr. 2024.

Pode ser usada para gerenciar uma ampla variedade de infraestrutura, como máquinas virtuais, balanceadores de carga, armazenamento e redes. Pode ser aplicada para criar e gerenciar máquinas virtuais em nuvens públicas e privadas, balanceadores de carga para distribuir o tráfego, volumes de armazenamento para máquinas virtuais, e redes para conectar máquinas virtuais.

Figura 9 – Um exemplo de fluxo Terraform



Fonte: Google Cloud; Arte/UT.

O fluxo de trabalho integrando Terraform e JavaScript para provisionar e gerenciar infraestrutura em nuvem é composto por seis etapas. Primeiramente, há a definição da infraestrutura desejada usando Terraform, através da criação de arquivos de configuração HCL que descrevem os recursos a serem provisionados. Em seguida, ocorre o planejamento, onde o Terraform gera um plano de execução detalhando as modificações a serem feitas na infraestrutura. Posteriormente, o plano é aplicado, provisionando ou atualizando os recursos na nuvem. Na quarta etapa, o estado da infraestrutura é sincronizado com o estado armazenado nos arquivos de configuração HCL. Ferramentas de notificação podem ser integradas na quinta etapa para enviar alertas sobre o status da execução do Terraform. Por fim, na sexta etapa, ocorre a integração com JavaScript para automatizar tarefas e adicionar funcionalidades extras, como ler e modificar arquivos de configuração, gerar configurações de infraestrutura

dinamicamente, executar comandos do Terraform e integrar com outras ferramentas e serviços. Considere a necessidade de provisionar uma instância de VM na nuvem. O Terraform pode ser usado para definir a configuração da VM, incluindo o tipo de instância, tamanho da memória, sistema operacional e configurações de rede. O JavaScript pode ser usado para ler um arquivo CSV com detalhes de várias VMs a serem provisionadas e gerar dinamicamente os arquivos de configuração HCL para cada VM. O Terraform então pode ser usado para provisionar todas as VMs automaticamente.

FINALIZANDO

Ao concluirmos este capítulo emocionante, espero que tenha sido possível absorver as bases fundamentais que delineiam o universo dos containers. Nossa jornada iniciou-se com uma exploração entusiasmada do Docker, sua inovação e aplicações revolucionárias. Ao longo do caminho, delimitamos as diferenças cruciais entre containers e máquinas virtuais, realçando a agilidade e eficiência inerentes aos containers no desenvolvimento e implementação de aplicações. Aprofundamos nosso entendimento ao mergulhar na orquestração de containers com o Kubernetes, uma ferramenta robusta que desvenda os desafios de gerenciar ambientes complexos. Além disso, desvendamos o poder transformador dos containers ao capacitar escalabilidade e distribuição, redesenhando a forma como concebemos, desenvolvemos e operamos *softwares* nos ambientes modernos. À medida que avançamos, fica evidente que os *containers* estão na vanguarda da revolução da computação em nuvem, moldando de maneira inovadora a entrega de soluções. Este capítulo serviu como seu guia confiável, proporcionando uma introdução perspicaz e abrindo caminho para as fascinantes possibilidades que os *containers* oferecem no cenário tecnológico atual. Estamos preparados para avançar, explorar e aproveitar plenamente o potencial dessa tecnologia dinâmica e transformadora.

REFERÊNCIAS

DOCKER overview. **Dockerdocs**, 2023. Disponível em: <https://docs.docker.com/engine/docker-overview/>. Acesso em: 16 abr. 2024.

FREEMAN, S. PRYCE, N. **Desenvolvimento de software orientado a objetos, guiado por testes**. Rio de Janeiro: Alta Books, 2012.

IZIDÓRIO, B. **O que é Docker?** 2019. Disponível em: <https://brunoizidorio.com.br/docker-o-que-e-docker/>. Acesso em: 05 nov. 2019.

SILBERSCHATZ, A.; GALVIN, P. **Sistemas Operacionais**. Rio de Janeiro: Campus, 2001.

TERKALY, B. Docker Container Overview for Business Leaders. **Microsoft**, 2016. Disponível em: <https://blogs.msdn.microsoft.com/allthingscontainer/2016/11/07/docker-container-overview-for-business-leaders/>. Acesso em: 16 abr. 2024.